IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Application of: Robert James Blainey et al.	:
	: Before the Examiner:
Serial No: 10/718,293	: Meng Yao Zhe
Filed: 11/20/2003	: Group Art Unit: 2195
Title: SOFTWARE BARRIER SYNCHRONIZATION	: Confirmation No.: 1209

APPELLANTS' BRIEF UNDER 37 C.F.R. §41.37

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This is an appeal to a final rejection dated July 18, 2008 of the claims in the Application. This brief is submitted pursuant to a Notice of Appeal filed on 10/20/2008.

BRIEF FOR APPLICANTS - APPELLANTS

(i)

Real Party in Interest

The real party in interest is International Business Machines Corporation (IBM), the assignee.

(ii)

Related Appeals and Interferences

There are no other appeals or interferences known to appellants, appellants' representative or assignee, which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(iii)

Status of Claims

Claims 1 - 31 were originally filed in the Application. Claims 9, 12, 13, 20 and 21 were subsequently canceled from the Application. Thus, Claims 1 – 8, 10, 11, 14 – 19 and 22 - 31 remain pending in the Application and are being appealed.

(iv)

Status of Amendment

An "Amendment-After-Final" was not filed.

(v)

Summary of Claimed Subject Matter

The invention, as claimed in independent Claim 1, provides a method of synchronizing N concurrently running processes in a data processing system at a first phase before allowing the N processes to proceed to a second phase, N >= 2. The method comprises: providing a first array of N elements initialized each to a first state wherein each i^{th} element of said first array is associated with an i^{th}

concurrently running process which will update the i^{th} element of the first array to a second state in response to completing the first phase, where 1 = i <= N (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); providing a second array of N elements initialized each to a hold state wherein each i^{th} element of said second array is associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is used to hold the i^{th} associated concurrently running process at the first phase, and is enabled to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase (see page 17, lines 1 – 18); and using a designated process configured to ascertain when the N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33).

The invention, as claimed in independent Claim 14, provides a system having at least one processor (any one of processors 11 in Fig. 1) for processing instructions to synchronize N concurrently running processes at a first phase before allowing the N processes to proceed to a second phase, where N >= 2. The instructions comprises instructions to use: a first array of N elements initialized each to a first state, each i^{th} element of said first array having an i^{th} concurrently running process associated therewith which will update the i^{th} element of the first array to a second state in response to completing the first phase, where 1 = i <= N (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); a second array of N elements with each element initialized to a hold state wherein each i^{th} element of said second array is associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is used to hold the i^{th} associated concurrently running process at the first phase, and is enabled to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase (see page 17, lines

1 – 18); a designated process configured to ascertain when the N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33).

The invention, as claimed in independent Claim 24, provides a processor (anyone of processors 11 in Fig. 1) for executing a process in order to synchronize said process at a phase with at least one other concurrently running process. The processor is operable to: access elements of a first array of elements, each accessed element of the first array being associated with one concurrently running process completing the phase and having an initial state (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); update an accessed element of said first array of elements to another state upon completion of the phase by said process, the accessed element being associated with the process (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); access an element of a second array of elements, said element of said second array being associated with said process wherein all elements of said second array have an initial hold state to hold an associated process at the phase and are enabled to switch, in response to receiving an instruction, to a release state to release the associated process to proceed to a next phase (see page 17, lines 1 – 18); and continually check said first array, using a designated process enabled to do so, to determine when all elements of the first array are updated in order to issue the instruction (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33).

The invention, as claimed in independent Claim 25, provides a method for executing a process in order to synchronize said process at a phase with at least one other concurrently running process. The method comprises: accessing elements of a first array of elements, each accessed element of the first array being associated with one concurrently running process completing the phase and having an initial state (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); updating an accessed element of said first array of

elements to another state upon completion of the phase by said process, the accessed element being associated with the process (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); accessing an element of a second array of elements, said element of said second array being associated with said process wherein all elements of said second array have an initial hold state to hold an associated process at the phase and are enabled to switch, in response to receiving an instruction, to a release state to release the associated process to proceed to a next phase (see page 17, lines 1 – 18); and continually checking said first array, using a designated process enabled to do so, to determine when all elements of the first array are updated to the other state in order to issue the instruction (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33).

The invention, as claimed in independent Claim 26, provides a processor (any one of processors 11 in Fig. 1) for executing a designated process in order to synchronize at least two concurrently running processes at a phase. The processor is operable to: access, using the designated process, an array of elements, each element of said array of elements being associated with one of said at least two concurrently running processes and having an initial state (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); monitor, using the designated process, all elements of said array of elements to detect when each of said elements of said array has been updated by its associated process (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33); hold, using the designated process, each one of the at least two concurrently running processes at the phase until an instruction to release the at least two concurrently running processes is received, the instruction indicating that the at least two concurrently running processes are synchronized at the phase (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33); and generate the instruction, using the designated process, when it is determined that all the elements of the array have been updated (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33).

The invention, as claimed in independent Claim 28, provides a computer program product for synchronizing N concurrently running processes in a data processing system at a first phase before allowing the N processes to proceed to a second phase, N >= 2. The computer program product comprises: a computer useable medium having computer readable program code means embodied in the medium for synchronizing the N concurrently running processes, the computer program code means including: computer readable program code means for providing a first array of N elements initialized each to a first state, each i^{th} element of said first array being associated with an i^{th} concurrently running process which will update the i^{th} element of the first array to a second state in response to completing the first phase, where 1 = i <= N (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); computer readable program code means for providing a second array of N elements initialized each to a hold state, each i^{th} element of said second array being associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is enabled to hold the i^{th} associated concurrently running process at the first phase and to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase see (page 17, lines 1 – 18); and computer readable program code means for monitoring said first array of N elements and, in response to each i^{th} element of said first array having had its state updated, generating said instruction for switching said N elements of said second array to said release state to allow the N processes to proceed to the second phase, said computer readable program monitoring code means including computer readable program code means for processing a designated process to monitor the first array of N elements and issue the generated instruction (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33). The code means of the claimed invention are the steps in figs. 3 – 6, algorithm 1 on page 15, algorithm 2 on page 19, and FORTRAN code on page 21.

The invention, as claimed in independent Claim 30, provides a system for synchronizing N concurrently running processes at a first phase before allowing the N processes to proceed to a second phase, N >= 2. The system comprises: means for providing a first array of N elements initialized each to a first state each i^{th} element of said first array being associated with an i^{th} concurrently running process which will update the i^{th} associated element of the first array to a second state in response to completing the first phase, where 1 = i <= N (see page 13, line 13 to page 14, line 9 and page 16, line 22 to page 17, line 1); means for providing a second array of N elements initialized each to a hold state, each i^{th} element of said second array being associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is enabled to hold the i^{th} associated concurrently running process at the first phase and is enabled to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase (see page 17, lines 1 – 18); and means for monitoring said first array of N elements and, upon each i^{th} element of said first array having had its state updated, generating said instruction for switching said N elements of said second array to said release state to allow the N processes to proceed to the second phase, said monitoring means including a process designated to monitor the first array of N elements to determine when the N elements of the first array are updated to the second state and to issue the generated instruction when processed by a processor (see page 17, line 19 to page 18, line 10 and, page 19, lines 26 - 33). The means of the claimed invention are the steps in figs. 3 – 6, algorithm 1 on page 15, algorithm 2 on page 19, and FORTRAN code on page 21 when processed by any one of processors 11 in Fig. 1.

(vi)

Grounds of Rejection to be Reviewed on Appeal

Whether it was proper to reject Claims 1 – 8, 10, 11, 14 – 19 and 22 - 31 under 35 U.S.C. §103(a) as being unpatentable over Ishihata et al. in view of Matsumoto

(vii)

Arguments

Whether it was proper to reject Claims 1 – 8, 10, 11, 14 – 19 and 22 - 31 under 35 U.S.C. §103(a) as being unpatentable over Ishihata et al. in view of Matsumoto

Claims 1, 14, 24 – 26, 28

In considering a §103 rejection, the subject matter of the claim "as a whole" must be considered and analyzed. In the analysis, it is necessary that the scope and contents of the prior art and differences between the art and the claimed invention (taken as a whole) be determined. *Graham v. John Deere Co.*, 383 U.S. 1 (1966).

The Examiner admitted that Ishihata et al. do not disclose the use of a designated process configured to ascertain when all N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase as claimed. However, the Examiner continued, Matsumoto teaches in col. 7, lines 35 – 48; Fig. 8, s11, s12, s9 the use of a designated process configured to ascertain when the N processes are all synchronized together to terminate the synchronization waiting state and proceed to the second phase by issuing a release instruction (i.e., whichever instruction that terminates the synchronization waiting state corresponds to the release instruction for the purpose of synchronization among multiple threads). Thus, the Examiner concluded, it would have been obvious for one skilled in the art to combine the teachings of Ishihata et al. with those of Matsumoto to arrive at the claimed invention. Appellants respectfully disagree.

Firstly, Ishihata et al. purport to teach a synchronization control system in a parallel computer. According to the teachings of Ishihata et al., the synchronization control system uses a synchronization request register, a synchronization detecting register, a status request register and a status detecting register for each processor of the parallel computer. The synchronization control system further uses two processing phases. The first processing phase is when the processors are actually processing data and the second processing stage is when the processors are processing messages. When a processor reaches a synchronization point, it requests synchronization by entering a logical "1" in its synchronization request register. When all the processors have done so, the "1s" in the synchronization request registers are ANDED and the result is used to reset the synchronization detecting register of each one of the processors.

When a processor finishes executing data, it will process a message. The message is sent from one processor to another. The message ensures that the system is not in error (i.e., no one processor is in error by being in a loop or driving a zero etc.). When a processor finishes handling a message and passes the message to the next processor, it sets its status request register by entering a logical "1" therein. As in the case of the synchronization request, when all the processors have set their status request register, the "1s" are ANDED and the result used to reset the status detecting register of the processors.

When both the synchronization detecting requests and the status detecting registers are (re)set, the system is synchronized and proceeds to the next processing stage.

Thus, Ishihata et al. use combinational logic technology to (1) determine when all the processes have completed a first phase and are to proceed to the next phase; and (2) to instruct the processes to proceed to the second stage when all the processes have completed the first stage.

Since Ishihata et al. provide a combinational logic system to determine when all the processes have completed a first phase and are to proceed to the

next phase and to instruct the processes to proceed to the second stage when all the processes have completed the first stage, there is no reason for anyone skilled in the art to combine the teachings of Ishihata et al. with those of Matsumoto in order to provide for ***a designated process configured to ascertain when the N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase*** as asserted by the Examiner.

Secondly, even if, arguendo, one skilled in the art would be inclined to combine the teachings of Ishihata et al. with those of Matsumoto, the combination would not teach, show or suggest the claimed invention.

Matsumoto teaches a method for synchronizing and scheduling multiple processes in a multiprocessor. According to Matsumoto, when one or more processes, each ineffectively waiting for synchronization, are dispatched to processors, they waste processor resources. As an example, let us suppose a plurality of processes are performing a certain operation in cooperation with one another (parallel processing) and during the operation, one or more processes are in a waiting state for synchronization. Let us further suppose that the number of processes is larger than the number of processors, so that all of the processes cannot be dispatched to the processors simultaneously. In this situation, if synchronization is achieved through a shared memory, the OS cannot determine whether or not a process, whose result is needed to synchronize the system, needs processing time. Therefore, depending on scheduling, only the processors waiting for synchronization will be dispatched. Consequently, the process whose result is needed to synchronize the system will not receive processing time. Thus, processor resources will continue to be wasted until the process whose result is needed to synchronize the system gets to be dispatched.

In such situations, Matsumoto discloses that the system has to be able to detect when all the processors are occupied with processes waiting for synchronization (i.e., when the system is in a busy loop waiting for synchronization). When in this busy loop, synchronization variables as well as

information related to system processor resources have to be checked. Depending on the result of the check: (1) the wait operation will be stopped and (2) control transferred to the OS which will call the scheduler to reschedule the processes. In that approach, the number of processes in an ineffective synchronization wait state can be reduced and the system processor resources more effectively used.

Matsumoto then proceeds to disclose a method for identifying when a system is in a busy loop in subsection 1.2, and a rescheduling method in subsection 1.3.

The Examiner cited passages in col. 7, lines 35 – 48, which is in subsection 1.2, and asserted that Matsumoto teaches the use of a designated process configured to ascertain when the N processes are all synchronized together to terminate the synchronization waiting state and proceed to the second phase by issuing a release instruction (i.e., whichever instruction that terminates the synchronization waiting state corresponds to the release instruction for the purpose of synchronization among multiple threads). Appellants disagree with the Examiner's assertion of what is disclosed in the cited passages.

In the paragraph in col. 7, lines 13 – 48, which explains Fig. 6 and parts of Fig. 8 which are identical to parts in Fig. 6 and which encompasses the passage cited by the Examiner, Matsumoto discloses:

> FIG. 7 shows a conventional loop for waiting. In FIG. 6, a synchronization variable is checked at the beginning of the procedure (S1) in order to incur as little overhead as possible as compared with the conventional method. The ideal case is one in which synchronization is established and the synchronization variable is set to a value established before checking the

synchronization variable a first time. If synchronization has completed, a waiting operation is immediately terminated. Only if the first time check indicates that synchronization has not completed, does the processor enter a synchronization waiting state (S2). In this state, variables to be affected by the entrance, for example, #MWC etc., are modified (S3). Information about system processor resources is read out (S4); determination is made of whether or not the process concerned should be terminated; and the scheduler is requested to reschedule processes depending on the above mentioned conditions (S5). If either of the conditions is fulfilled, variables to be affected are modified (S6), and the scheduler is invoked using a system call for transferring control of the processor and the like (S7). If neither of the conditions is fulfilled, synchronization variables are newly checked (S11). If synchronization is not established, operation returns to the read-out step of processor resource information and the procedure is iterated. If synchronization is established, the processor concerned completes the synchronization waiting state, and then affected variables (#MWC etc.,) are modified (S12), and the waiting operation is

> terminated (S9). When the process which has transferred control of the processor is again dispatched to that processor or another processor, the operation joins the flow taken when neither of the conditions for process switching is fulfilled (S8, S10).

According to the above-reproduced paragraph and Fig. 8, when a processor is to execute a process, it first checks to see whether the system is synchronized (step S_1). If the system is synchronized, the processor does not put the process into a synchronization waiting state. If the system is not synchronized then the processor puts the process into the synchronization waiting state (step S_2). The processor then adds the process to the number of processes which are in the synchronization waiting state (step S_3 of Fig. 8) and reads information regarding processor resources (step S_4). Then, the processor determines whether the scheduler should be called to reschedule the processes in the system (step S_5). If so, the processor takes the process out of the number of processes in the synchronization waiting state (step S_6) and makes a system call for process switching (step S_7). After doing so, the processor again checks to see whether the system is synchronized (step S_8). If the system is still not synchronized, then the processor adds the process back to the number of processes in the synchronization waiting state (step S_{10}) and goes back to step S_4. If, on the other hand, the system is now synchronized, the processor terminates the process' synchronization waiting state (step S_9).

If in step S_5 it was determined that the scheduler should not be called to reschedule the processes, the processor will check again to see whether the system is synchronized (step S_{11}). If the system is still not synchronized then the processor goes back to step S_4. If the system is now synchronized, the processor takes the process out of the number of processes in the

synchronization waiting state (step S_{12}) and terminates the synchronization waiting state (step S_9).

Note that each processor processing a process will go through the process in the flowchart in Fig. 8 (see steps S_1 and S_2 as well as adding and subtracting the process from the number of processes in the synchronization waiting state). Consequently, each processor will have to terminate the synchronization waiting state of each process that it is processing when the system becomes synchronized.

Thus in the above-reproduced paragraph, Matsumoto discloses that when a processor is to execute a process, the processor first determines whether or not it needs to put the process in a synchronization waiting state (based on whether or not the system is already synchronized) and/or terminates the synchronization waiting state of the process (when the system becomes synchronized) in the case where the process was in a synchronization waiting state. As mentioned above, putting the process in a synchronization waiting state when the system is not synchronized frees the processor up to reschedule the processes ins the system such that a process that is holding up the synchronization of the system can quickly receive processing time in order to achieve system synchronization.

Therefore, combining the teachings of Ishihata et al. with those of Matsumoto teaches a combinational logic technology that (1) determines when all the processes have completed a first phase and are to proceed to the next phase; and (2) instructs the processes to proceed to the second stage when all the processes have completed the first stage wherein each processor that is to execute a process first determines whether or not to put the process in a synchronization waiting state (based on whether or not the system is already synchronized) and/or terminates the processor's synchronization waiting state (when the system becomes synchronized) in the case where the process is in a synchronization waiting state.

But, the combination of the two references does not teach, show or suggest using ***a designated process configured to ascertain when all N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase*** as asserted by the Examiner. This is especially so, since as asserted by the Examiner and disclosed by Matsumoto, any one of the processes of Matsumoto may be the last process whose synchronization waiting state is terminated. By definition therefore, that process cannot be a ***designated*** process.

Hence, Appellants submit that the claims are patentable over the combination of the teachings of Ishihata et al. and Matsumoto.

Claim 8

Claim 8 includes the limitations "wherein the designated process is not one of the N concurrently running processes."

The Examiner asserted that Ishihata et al. teach the limitations of Claim 8 in col. 10, lines 49 – 54. Appellants respectfully disagree.

Firstly, in rejecting the independent claims the Examiner admitted that Ishihata et al. do not disclose the use of ***a designated process configured to ascertain when all N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase*** and that it is Matsumoto that does so. Therefore, the Examiner is being contradictory in now asserting that Ishihata et al. teach the ***designated process*** as not being one of the N concurrently running processes.

Secondly, in col. 10, lines 49 – 54, Ishihata et al. disclose:

> A CPU of each PE can be informed that synchronization is established by monitoring the synchronization detecting register 23 through a dynamic loop and by an interrupt due to the synchronization detecting register 23 being set. The CPU which detects a synchronization proceeds to the next step.

In the above-reproduced paragraph, Ishihata et al. teach that a CPU monitors a synchronization register to determine when the system is synchronized and to proceed to the next step.

However, Ishihata et al. do not teach, show or so much as suggest in this paragraph a ***designated process*** much less a ***designated process*** that is not one of the N concurrently running processes as asserted by the Examiner.

Consequently, Appellants submit that Claim 8 is patentable over the applied references.

In view of the foregoing, Appellants kindly request withdrawal of the rejection and passage to issue of the claims in the Application.

Respectfully Submitted

By: _____

Volel Emile
Attorney for Applicants
Registration No. 39,969
(512) 306-7969

(viii)

Claims Appendix

1. A method of synchronizing N concurrently running processes in a data processing system at a first phase before allowing the N processes to proceed to a second phase, N >= 2, comprising:

providing a first array of N elements initialized each to a first state wherein each i^{th} element of said first array is associated with an i^{th} concurrently running process which will update the i^{th} element of the first array to a second state in response to completing the first phase, where 1 = i <= N;

providing a second array of N elements initialized each to a hold state wherein each i^{th} element of said second array is associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is used to hold the i^{th} associated concurrently running process at the first phase, and is enabled to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase; and

using a designated process configured to ascertain when the N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase.

2. The method recited in claim 1, wherein each i^{th} process of said N concurrently running processes waits at its i^{th} associated element of said second array for said release state in response to completing the first phase and updating the i^{th} associated element of the first array.

3. The method recited in claim 2, wherein each i^{th} element of said first array has a byte size corresponding to a size of a cache line used in said data processing system.

4. The method recited in claim 3, wherein each i^{th} element of said second array has a byte size corresponding to the size of said cache line used in said data processing system.

5. The method recited in claim 4, wherein each i^{th} element of said second array is provided locally in relation to its i^{th} respective, associated process.

6. The method recited in claim 2, wherein after the N elements of said first array are updated to the second state, and prior to issuance of the release instruction the N elements of said first array are reinitialized to the first state.

7. The method recited in claim 1, wherein the designated process is one of said N concurrently running processes.

8. The method recited in claim 1, wherein the designated process is not one of the N concurrently running processes.

9. Canceled.

10. The method recited in claim 1, wherein each i^{th} element of said first array and said second array comprises a state machine.

11. The method recited in claim 10, wherein said state machine is one of a counter, a gate, a flag and a sensor.

12. Canceled.

13. Canceled.

14. A system having at least one processor for processing instructions to synchronize N concurrently running processes at a first phase before allowing the N processes to proceed to a second phase, where N >= 2, the instructions comprising instructions to use:

a first array of N elements initialized each to a first state, each i^{th} element of said first array having an i^{th} concurrently running process associated therewith which will update the i^{th} element of the first array to a second state in response to completing the first phase, where 1 = i <= N;

a second array of N elements with each element initialized to a hold state wherein each i^{th} element of said second array is associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is used to hold the i^{th} associated concurrently running process at the first phase, and is enabled to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase;

a designated process configured to ascertain when the N elements of the first array are updated to the second state to issue the release instruction to allow the N processes to proceed to the second phase.

15. The system recited in claim 14, wherein each i^{th} element of said first array has a byte size corresponding to a size of a cache line used in said data processing system.

16. The system recited in claim 15, wherein each i^{th} element of said second array has a byte size corresponding to the size of said cache line used in said data processing system.

17. The system recited in claim 14, wherein each i^{th} element of said second array is provided locally in relation to its respective, i^{th} associated process.

18. The system recited in claim 14, wherein each i^{th} element of said first array and said second array is a state machine.

19. The system recited in claim 14, wherein said state machine is one of a counter, a gate, a flag and a switch.

20. Canceled.

21. Canceled.

22. The system recited in claim 14, wherein said N concurrently running processes execute on multiple processors embodied within a single computer.

23. The system recited in claim 14, wherein said N concurrently running processes execute on multiple processors distributed across multiple computers connect across a network.

24. A processor for executing a process in order to synchronize said process at a phase with at least one other concurrently running process, said processor being operable to:

access elements of a first array of elements, each accessed element of the first array being associated with one concurrently running process completing the phase and having an initial state;

update an accessed element of said first array of elements to another state upon completion of the phase by said process, the accessed element being associated with the process;

access an element of a second array of elements, said element of said second array being associated with said process wherein all elements of said second array have an initial hold state to hold an associated process at the phase and are enabled to switch, in response to receiving an instruction, to a release state to release the associated process to proceed to a next phase; and

continually check said first array, using a designated process enabled to do so, to determine when all elements of the first array are updated in order to issue the instruction.

25. A method for executing a process in order to synchronize said process at a phase with at least one other concurrently running process, comprising:

accessing elements of a first array of elements, each accessed element of the first array being associated with one concurrently running process completing the phase and having an initial state;

updating an accessed element of said first array of elements to another state upon completion of the phase by said process, the accessed element being associated with the process;

accessing an element of a second array of elements, said element of said second array being associated with said process wherein all elements of said second array have an initial hold state to hold an associated process at the phase and are enabled to switch, in response to receiving an instruction, to a release state to release the associated process to proceed to a next phase; and

continually checking said first array, using a designated process enabled to do so, to determine when all elements of the first array are updated to the other state in order to issue the instruction.

26. A processor for executing a designated process in order to synchronize at least two concurrently running processes at a phase, said processor being operable to:

access, using the designated process, an array of elements, each element of said array of elements being associated with one of said at least two concurrently running processes and having an initial state;

monitor, using the designated process, all elements of said array of elements to detect when each of said elements of said array has been updated by its associated process;

hold, using the designated process, each one of the at least two concurrently running processes at the phase until an instruction to release the at least two concurrently running processes is received, the instruction indicating that the at least two concurrently running processes are synchronized at the phase; and

generate the instruction, using the designated process, when it is determined that all the elements of the array have been updated.

27. The processor recited in claim 26, wherein said designated process is one of said concurrent processes.

28. A computer program product for synchronizing N concurrently running processes in a data processing system at a first phase before allowing the N processes to proceed to a second phase, N >= 2, the computer program product comprising:

a computer useable medium having computer readable program code means embodied in the medium for synchronizing the N concurrently running processes, the computer program code means including:

computer readable program code means for providing a first array of N elements initialized each to a first state, each i^{th} element of said first array being associated with an i^{th} concurrently running process which will update the i^{th} element of the first array to a second state in response to completing the first phase, where 1 = i <= N;

computer readable program code means for providing a second array of N elements initialized each to a hold state, each i^{th} element of said second array being associated with the i^{th} concurrently running process associated with the i^{th} element of the first array and is enabled to hold the i^{th} associated concurrently running process at the first phase and to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase; and

computer readable program code means for monitoring said first array of N elements and, in response to each i^{th} element of said first array having had its state updated, generating said instruction for switching said N elements of said second array to said release state to allow the N processes to proceed to the second phase, said computer readable program monitoring code means including computer readable program code means for processing a designated process to monitor the first array of N elements and issue the generated instruction.

29. The computer program product recited in claim 28, wherein each i^{th} process of said N concurrently running processes waits at its i^{th} associated element of said second array for said release state in response to completing the first phase and updating the i^{th} associated element of the first array.

30. A system for synchronizing N concurrently running processes at a first phase before allowing the N processes to proceed to a second phase, N >= 2, comprising:

means for providing a first array of N elements initialized each to a first state each i^{th} element of said first array being associated with an i^{th} concurrently running process which will update the i^{th} associated element of the first array to a second state in response to completing the first phase, where 1 = i <= N;

means for providing a second array of N elements initialized each to a hold state, each i^{th} element of said second array being associated with the i^{th} concurrently running process associated with the i^{th} element of the first

array and is enabled to hold the i^{th} associated concurrently running process at the first phase and is enabled to switch, in response to receiving a release instruction, to a release state to release the i^{th} associated concurrently running process to proceed to the second phase; and

means for monitoring said first array of N elements and, upon each i^{th} element of said first array having had its state updated, generating said instruction for switching said N elements of said second array to said release state to allow the N processes to proceed to the second phase, said monitoring means including a process designated to monitor the first array of N elements to determine when the N elements of the first array are updated to the second state and to issue the generated instruction when processed by a processor.

31. The system recited in claim 30, wherein each i^{th} process of said N concurrently running processes waits at its i^{th} associated element of said second array for said release state in response to completing the first phase and updating the i^{th} associated element of the first array.

(ix)

Evidence Appendix

None.

(x)

Related Proceedings Appendix

None.